Satisfiability of XPath on data trees

Diego Figueira

Satisfiability of XPath on data trees

Diego Figueira

CNRS & LaBRI, France

Abstract

This is a survey on the satisfiability problem for XPath on data trees. *Data trees* are finite trees whose every node carries a label from a finite alphabet and a data value from an infinite domain. XPath is an expressive node selecting language for XML documents, which can be abstracted as data trees. Its satisfiability problem is in general undecidable. However, various fragments of XPath have decidable satisfiability problem, these are fragments defined in terms of the navigational axes which are allowed. We explore the state of the art in terms of decidability and discuss briefly some of the algorithmic techniques involved.

1 Introduction

A data tree is a tree whose every node contains a label from a finite alphabet and a data value from some infinite domain. This model has been considered as a suitable abstraction for semistructured data such as XML documents: XML tags are modelled as data tree labels, and XML attributes as its data values. A simpler abstraction would only take into account the tags, resulting in a tree over a finite alphabet, a model that has been intensively studied in automata theory. However, this abstraction ignores all actual data stored in the XML document attributes. This is why there has been an interest in 'data' trees as trees that also carry data from an infinite domain. Data trees have also been considered in the realm of timed automata, program verification, and generally in systems manipulating data values. Finding decidable logics or automata models over data trees is generally a relevant question when studying data-driven systems.

More precisely, a *data tree* is an unranked ordered finite tree whose every node contains a label and a data value. Labels belong to some finite alphabet (*e.g.*, the set $\{a, b\}$), and data values to some infinite domain (*e.g.*, $\mathbb{N} = \{0, 1, 2, \dots\}$), see Figure 1 for an example. In this paper we survey the decidability status on a family of logics based on XPath, which is arguably the most widely used XML query language. XPath is implemented in XSLT and XQuery and it is used as a

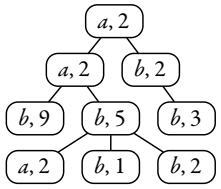

Fig. 1: A data tree over $\mathbb{A}\times\mathbb{D}$, whose alphabet and data domain are $\mathbb{A} = \{a, b\}$ and $\mathbb{D} = \mathbb{N}$.

constituent part of several specification and update languages. XPath is fundamentally a general purpose language for addressing, searching, and matching pieces of an XML document. It is an open standard and constitutes a World Wide Web Consortium (W3C) Recommendation [CD99].

Query containment and query equivalence are fundamental static analysis problems, which can be useful to, for example, query optimization tasks. In logics closed under boolean operators—as the ones treated here— these problems reduce to checking for *satisfiability*: Is there a document on which a given query has a non-empty result? By answering this question we can decide at compile time whether the query contains a contradiction and thus the computation of the query (or subquery) on the document can be avoided. Or, by answering the query equivalence problem, one can test if a query can be safely replaced by another one which is more optimized in some sense (*e.g.*, in the use of some resource). Moreover, the satisfiability problem is crucial for applications on security [FCG04], type checking transformations [MN07], and consistency of XML specifications.

Core-XPath (term coined in [GKP05]) is a fragment of XPath 1.0 that captures all the navigational behavior of XPath. It has been well studied and its satisfiability problem is known to be decidable even in the presence of DTDs. The extension of this language with the possibility to make equality and inequality tests between attributes of elements in the XML document is named Core-Data-XPath in [Boj+09].

In a nutshell, the important formulas of Core-Data-XPath (henceforth *XPath*) are of the form

$$\langle \alpha = \beta \rangle,$$

where α, β are *path expressions*, that navigate the tree using modalities, or *axes*, and which can make tests at intermediary nodes. Available axes are the natural navigation relations on a tree: child (that we note $\downarrow$), descendant ($\downarrow_*$), ancestor ($\uparrow^*$), next-sibling ($\rightarrow$), etc. Such a formula $\langle \alpha = \beta \rangle$ is *true* at a node x of a data tree if there are two nodes y, z in the tree that can be reached with the relations denoted by α, β respectively, so that they both carry the *same* datum. Analogously, a formula of the form $\langle \alpha \neq \beta \rangle$ tests that one can reach *different* data values. For example, the formula $\langle \downarrow\downarrow[a] = \uparrow^*[b] \rangle$ holds true at any node of a data tree having a grandchild labelled a carrying the same data value as an ancestor labelled b; the formula $\neg\langle \downarrow_*[a] \neq \downarrow_*[a] \rangle$ states that all the a-labelled descendants carry the same data value; and a more complex formula like $\neg\langle \uparrow^*[\neg\langle\uparrow\rangle] \neq \uparrow^*\downarrow_*[\neg\langle\downarrow\rangle] \rangle$ states that all the leaves carry the same data value as the root (and this holds irrespectively of the node where the formula is evaluated).

Other formalisms for trees with data values

Several formalisms have been studied in relation to static analysis on trees with data values. We mention here some of the most prominent examples.

First-order logic. One possible formalism is $\mathrm{FO}(<_h, succ_h, <_v, succ_v, \sim)$, first order logic with binary relations to navigate the tree: the descendant $<_v$, child $succ_v$, right sibling $succ_h$ and following sibling $<_h$ (*i.e.*, the transitive closure of $succ_h$); and an equivalence relation $\sim$ to express that two nodes of the trees have the same data value. However, the satisfiability problem is undecidable, even when restricted to using only three variables, and even on data words [Boj+10]. When restricted to using only two variables, the decidability status for the satisfiability of $\mathrm{FO}^2(<_h, succ_h, <_v, succ_v, \sim)$ remains unknown, although it is known to be at least as hard as the reachability problem for BVAS (Branching Vector Addition System) [Boj+09], a well-known open problem in verification, whose decidability is only known for dimension one [Göl+16; Fig+17]. However, if the signature has only the child and next-sibling relation —$\mathrm{FO}^2(succ_h, succ_v, \sim)$— the logic

is decidable in 3NExpTime [Boj+09]. Also, on the class of bounded-depth data trees $\mathrm{FO}^2(<_h, succ_h, <_v, succ_v, \sim)$ becomes decidable [BB07] —a result that can be seen as a non-trivial extension of the decidability of the logic on data words [Boj+10].

Automata. There has also been works on automata models for trees with data. Tree automata with registers to store and compare data values were studied in [KT08] as an extension to a similar model on words [KF94; NSV04]. A decidable alternating version of these automata called ATRA was studied in [JL11], and it was extended in [Fig10; Fig12a] to show decidability of the satisfiability problem for a fragment of XPath (discussed in §4.2). The work [BL10] introduces a simple yet powerful automata model called *Class Automata* on data trees that can capture $\mathrm{FO}^2(<_h, succ_h, <_v, succ_v, \sim)$, XPath, ATRA, and other models. Although its emptiness problem is undecidable, classes of data trees for which it is decidable are studied in [Bár+12].

Modal logics. One can also consider extensions of modal logics such as CTL or μ-calculus with a data binding mechanism for storing and comparing the data values that have been seen along the structure during the evaluation of the formula [DLN05; JL07; KSZ10]. Here we concentrate on XPath, which is incomparable in terms of expressiveness with all the previously mentioned formalisms (except for Class Automata). While XPath has a modal flavour, it does not have a data binding operator, instead it uses the aforementioned formulas $\langle \alpha = \beta \rangle$ to test for the existence of paths leading to nodes with equal data. Finally, another logic that can be seen as a weak version of XPath is the so-called "logic of repeating values", or LRV, studied in [DDG07; DFP16; AFF17]. LRV can be seen as the common factor between the data binding approach and XPath.

Other formalisms include tree automata combined with set and linear constraints on cardinalities of sets of data values [DLT12; Tan12], Datalog programs [Abi+13], or the problem of containment of pattern-based queries [Dav+13; FL14; ACK11] and positive XPath fragments [NS06; DT01].

Focus of this overview. Here we concentrate on the *finite satisfiability problem* for XPath, with a stress on algorithmic techniques required by the presence of data values, negation, and transitive axes. For an overview on the expressiveness and evaluation of XPath, we refer the reader to [BK08; BP11]. The problem of testing for bisimulation in this setting has also been addressed in [Abr+16; FFA15]. Axiomatizations and calculi for XPath have been proposed in [BLS16; Abr+17; AFS17; AF16].

2 Preliminaries

As usual, we use $\cdot^+$, $\cdot^*$, $\cdot^{-1}$ and $\circ$ to denote, respectively, the transitive closure, the reflexive-transitive closure, the inverse and the composition of binary relations.

Unranked finite trees. By $\mathit{Trees}(\mathbb{A} \times \mathbb{D})$ we denote the set of finite ordered and unranked data trees over a finite alphabet $\mathbb{A}$ and data domain $\mathbb{D}$. We use the letter $\mathcal{T}$ to denote a data tree, and T to denote the set of its nodes. For each node $x \in T$ we refer by $\mathit{label}_{\mathcal{T}}(x) \in \mathbb{A}$ to its label and by $\mathit{data}_{\mathcal{T}}(x) \in \mathbb{D}$ to its data value. The *right-sibling* of a node is the first sibling to the right of the node, should it exist, while the *left-sibling* is the one to the left.

3 XPath on data trees

3.1 Definition

XPath is a two-sorted language, with *path* expressions (that we write α, β) and *node* expressions (that we write φ, ψ). Path expressions express properties of pairs of nodes in a data tree, and thus

Syntax:

$$\begin{array}{rll} \textit{path expressions} & \alpha,\beta & ::= \varepsilon \mid \sigma \mid \sigma^* \mid \sigma^+ \mid [\varphi] \mid \alpha\beta \qquad \text{for } \sigma \in \{\downarrow,\uparrow,\rightarrow,\leftarrow\}, \\ \textit{node expressions} & \varphi,\psi & ::= a \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle\alpha\rangle \mid \langle\alpha = \beta\rangle \mid \langle\alpha \neq \beta\rangle \end{array}$$

Semantics:

$$\textit{path expressions} \begin{cases} [\![\varepsilon]\!]^{\mathcal{T}} = \{(x,x) \mid x \in T\} \\ [\![\downarrow]\!]^{\mathcal{T}} = \{(x,y) \in T \times T \mid y \text{ is a child of } x\} \\ [\![\uparrow]\!]^{\mathcal{T}} = ([\![\downarrow]\!]^{\mathcal{T}})^{-1} \\ [\![\rightarrow]\!]^{\mathcal{T}} = \{(x,y) \in T \times T \mid y \text{ is the right-sibling of } x\} \\ [\![\leftarrow]\!]^{\mathcal{T}} = ([\![\rightarrow]\!]^{\mathcal{T}})^{-1} \\ [\![\sigma^*]\!]^{\mathcal{T}} = ([\![\sigma]\!]^{\mathcal{T}})^* \qquad \text{for } \sigma \in \{\downarrow,\uparrow,\rightarrow,\leftarrow\} \\ [\![\sigma^+]\!]^{\mathcal{T}} = ([\![\sigma]\!]^{\mathcal{T}})^+ \qquad \text{for } \sigma \in \{\downarrow,\uparrow,\rightarrow,\leftarrow\} \\ [\![\,[\varphi]\,]\!]^{\mathcal{T}} = \{(x,x) \in T \times T \mid x \in [\![\varphi]\!]^{\mathcal{T}}\} \\ [\![\alpha\beta]\!]^{\mathcal{T}} = [\![\alpha]\!]^{\mathcal{T}} \circ [\![\beta]\!]^{\mathcal{T}} \end{cases}$$

$$\textit{node expressions} \begin{cases} [\![a]\!]^{\mathcal{T}} = \{x \in T \mid \mathit{label}_{\mathcal{T}}(x) = a\} \\ [\![\neg\varphi]\!]^{\mathcal{T}} = T \setminus [\![\varphi]\!]^{\mathcal{T}} \\ [\![\varphi \wedge \psi]\!]^{\mathcal{T}} = [\![\varphi]\!]^{\mathcal{T}} \cap [\![\psi]\!]^{\mathcal{T}} \\ [\![\varphi \vee \psi]\!]^{\mathcal{T}} = [\![\varphi]\!]^{\mathcal{T}} \cup [\![\psi]\!]^{\mathcal{T}} \\ [\![\langle\alpha\rangle]\!]^{\mathcal{T}} = \{x \in T \mid \exists y \in T \text{ s.t. } (x,y) \in [\![\alpha]\!]^{\mathcal{T}}\} \\ [\![\langle\alpha{=}\beta\rangle]\!]^{\mathcal{T}} = \{x \in T \mid \exists y,z \text{ s.t. } (x,y) \in [\![\alpha]\!]^{\mathcal{T}}, (x,z) \in [\![\beta]\!]^{\mathcal{T}}, \mathit{data}_{\mathcal{T}}(y) = \mathit{data}_{\mathcal{T}}(z)\} \\ [\![\langle\alpha{\neq}\beta\rangle]\!]^{\mathcal{T}} = \{x \in T \mid \exists y,z \text{ s.t. } (x,y) \in [\![\alpha]\!]^{\mathcal{T}}, (x,z) \in [\![\beta]\!]^{\mathcal{T}}, \mathit{data}_{\mathcal{T}}(y) \neq \mathit{data}_{\mathcal{T}}(z)\} \end{cases}$$

Fig. 2: The syntax of XPath and its semantics on a data tree $\mathcal{T}$.

denote a binary relation. These relations result from composing the child, parent, right- and left-sibling, etc., with the possibility of testing for node expressions. Node expressions denote properties of nodes. For example, they can test that the node has a certain label; or that it has a child labeled a with the same data value as an ancestor labeled b, which is expressed by $\langle\downarrow[a] = \uparrow^*[b]\rangle$. As another example, we can select the nodes that have an a-labeled left sibling with the same data as some descendant of a right sibling with the formula $\varphi = \langle {}^*\!\leftarrow[a] = \rightarrow^*\downarrow_*\rangle$. Its formal syntax and semantics are defined in Figure 2. A *formula* of XPath is either a node expression or a path expression. We will refer to different *navigational fragments* of XPath, with the notation XPath(Σ), for any $\Sigma \subseteq \{\downarrow,\downarrow_*,\downarrow_+,\uparrow,\uparrow^*,\uparrow^+,\rightarrow,\rightarrow^*,\rightarrow^+,\leftarrow,{}^*\!\leftarrow,{}^+\!\leftarrow\}$, as those formulas containing only axes from Σ. One can also consider an extension of XPath(Σ) by the possibility of adding the Kleene star to any path expression. We call this extension regXPath(Σ) (for *regular*-XPath(Σ)). Note that, for example, in terms of expressive power regXPath($\downarrow$) = regXPath($\downarrow,\downarrow_*$) $\supsetneq$ XPath($\downarrow,\downarrow_*$); indeed regXPath($\downarrow$) can test that there exists a downward path of a-labelled nodes until a b-node is reached through $\langle(\downarrow[a])^*\downarrow[b]\rangle$, while XPath($\downarrow,\downarrow_*$) cannot express this property. More generally, regXPath can express any regular property of paths, such as paths with an even number of a's:

$$\big((\downarrow[\neg a])^*\downarrow[a](\downarrow[\neg a])^*\downarrow[a](\downarrow[\neg a])^*\big)^*.$$

3.2 Expressive power

We give some examples of the properties that XPath can express when evaluated at the root node of a data tree.

(1) **Key.** The property "all nodes labelled with label a have different data values", which we call *key* property, can be expressed in XPath($\downarrow_+, \rightarrow^+$) by negating both the property[1]

$$\langle \downarrow_* [a \wedge \langle \varepsilon = \downarrow_+ [a] \rangle] \rangle$$

asserting that there are two nodes in ancestor-descendant relation contradicting the Key property; and the property

$$\langle \downarrow_+ [\langle \downarrow_* [a] = \rightarrow^+ \downarrow_* [a] \rangle] \rangle$$

describing that there are two nodes at incomparable positions contradicting the Key property. However, this property cannot be expressed in other fragments such as XPath($\downarrow_*, \rightarrow^*$) or XPath($\downarrow, \downarrow_+, \rightarrow$).

(2) **Foreign key.** The property "for every a-labelled node there is a b-labelled node with the same data value" can be easily expressed in XPath($\downarrow_*, \uparrow^*$) by negating $\langle \downarrow_* [a \wedge \neg \langle \varepsilon = \uparrow^* \downarrow_* [b] \rangle] \rangle$.

(3) **Matching.** One can also express "for every a-labelled node there is *exactly one* b-node with the same data value" by mixing the key and foreign key properties. This kind of properties lies at the basis of the proofs for undecidability or very high (non-primitive recursive) lower bounds for the satisfiability of some of these fragments.

3.3 Relation to FO

In terms of expressive power, XPath (with all its axes) contains all of $\mathrm{FO}^2(<_h, succ_h, <_v, succ_v, \sim)$. This containment is strict as $\mathrm{FO}^2(<_h, succ_h, <_v, succ_v, \sim)$ cannot express properties of the form "every a-labelled node x has a b-labelled descendant y with the same data value so that there is a c-labelled node between x and y" using only two variables, while this can be expressed through $\neg \langle \downarrow_* [\neg \langle [a] = \downarrow_* [c] \downarrow_* [b] \rangle] \rangle$ in XPath. However, as soon as we don't count with horizontal navigation, these logics become incomparable in expressive power. Indeed, XPath($\downarrow, \downarrow_*, \uparrow, \uparrow^*$) can still express the property just stated, while it cannot express the Key property (1), which can be expressed in $\mathrm{FO}^2(\sim)$, even without the use of any navigational relation: $\forall x \forall y \; a(x) \wedge a(y) \wedge x \neq y \rightarrow \neg(x \sim y)$. Intuitively, this is because XPath($\downarrow, \downarrow_*, \uparrow, \uparrow^*$) lacks the power to navigate to incomparable positions, and thus the ability to make sure that it navigates to a different node. These examples are in contrast with the fact that in the absence of data values XPath and $\mathrm{FO}^2(<_h, succ_h, <_v, succ_v, \sim)$ are equally-expressive.

4 Satisfiability of XPath

The satisfiability problem for XPath, or SAT-XPath, is the problem of whether, given a node expression φ of XPath, there exists a data tree $\mathcal{T}$ over which the semantics of the formula is non empty at at its root, in which case we write $\mathcal{T} \models \varphi$.

[1] Note that $\downarrow_*$ can be expressed using $\downarrow_+$, although the reciprocal doesn't hold.

The satisfiability problem for XPath is undecidable [GF05] already on data words (*i.e.*, rank 1 trees). How can we regain decidability for satisfiability of XPath then? We can restrict the models, or restrict the logic. The first possibility is to restrict the classes of documents on which we evaluate the query [Bár+12; BB07]. Another, more studied, approach is to restrict the syntax. One way to regain decidability is to syntactically restrict the amount of nodes that XPath properties can describe. In this vein, there have been thorough studies on many fragments without negation or without transitive axes [BFG08; GF05]. These fragments enjoy a small model property and are therefore decidable, with complexities ranging between PTime and NExpTime depending on the use we allow of negation, transitive closure and nesting. However, they cannot express *global* properties, involving possibly the values of all the nodes in an XML document. In this survey we consider the following desirable features on a logic

- closed under boolean operators,
- having as much freedom as possible to navigate the tree in many directions: up, down, left, right,
- having the possibility to reach any node of the tree, with transitive axes, like descendant, following sibling (the transitive closure of the next sibling axis), etc.

In the next sections we discuss several results of the last 10 years regarding navigational fragments which allow: only to go downward, to go downward and rightwards, to go downward and upward. As we will see, all these fragments have a decidable satisfiability problem. Finally, we will discuss some (un)decidability results that reveal the surprising difference between using transitive axes and using *reflexive*-transitive axes in this context.

4.1 Downward fragment

Downward XPath is the fragment of XPath using only child and descendant relations, that is XPath($\downarrow, \downarrow_*$). Its satisfiability problem is known to be decidable,

Theorem 1 ([Fig12b])**.**

- *SAT*-XPath($\downarrow, \downarrow_*$) *and SAT*-XPath($\downarrow_*$) *are* ExpTime*-complete;*
- *SAT*-XPath($\downarrow$) *is* PSpace*-complete.*

The satisfiability problem is shown via a reduction to an finite state automata model, which implies that the same complexities remain if we replace XPath with regXPath [Fig12b].

The key property (1) enjoyed by XPath($\downarrow, \downarrow_*$) used in the decidability proof of the theorem above is the property of closure under subtree replication. For every tree one can duplicate a subtree as a sibling without changing the semantics of any formula at any node of the tree; as depicted in Figure 3, this means that it is safe to make the witness tree to grow 'fatter'. This allows to have enough space to find exponential size witnessing paths for every path expression. If this property would fail, for example as a result of working with *ranked* data trees, one would obtain a non-primitive recursive lower bound as it will be discussed later. A corollary of the proof also shows that for every satisfiable formula of XPath($\downarrow, \downarrow_*$) there exists a witnessing data tree so that every pair of incomparable subtrees share only a polynomial number of data values (polynomial in the formula).

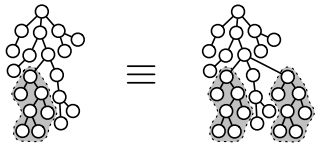

Fig. 3: Subtree replication.

$\downarrow$	$\downarrow_*$	**Complexity**
•		PSPACE-complete
	•	EXPTIME-complete
•	•	EXPTIME-complete

Tab. 1: Satisfiability for downward XPath. All complexity results hold for regXPath as well.

Interestingly, while SAT-XPath($\downarrow_*$) is EXPTIME-complete, this complexity drops to PSPACE if ε expressions are forbidden [Fig12b, Proposition 6.15], that is, the fragment where all data tests are of the form $\langle \downarrow_* \alpha = \downarrow_* \beta \rangle$ or $\langle \downarrow_* \alpha \neq \downarrow_* \beta \rangle$ for some α, β. Another fragment enjoying a PSPACE complexity is the fragment whose data tests are of the form $\langle \varepsilon = \downarrow_* [\varphi] \rangle$ [BLS16, Theorem 21]. This is because in both fragments every satisfiable formula is satisfied by a data tree of polynomial depth, *i.e.*, they enjoy the *poly-depth model property*. In fact, it is not hard to see that *any* fragment of regXPath($\downarrow, \downarrow_*$) with the poly-depth model property is necessarily in PSPACE [Fig12b, Proposition 6.7] —in particular, XPath($\downarrow$). On the other hand, as soon as we can use ε in data tests and we count with the possibility of composing two downward tests, as in formulas of the form $\langle \varepsilon = \downarrow_* [\varphi] \downarrow_* [\psi] \rangle$, one can force a model to have exponential height [Fig12b, Theorem 6.5].

4.2 Forward fragment

The forward fragment of XPath is an extension of the previous fragment using the child, descendant, right-sibling and following-sibling axes, that is XPath($\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$). While its satisfiability problem is still decidable, the complexity is gigantic: non-primitive recursive hard (NPR). That is, there is no algorithm that can decide the satisfiability problem using space or time bounded by a primitive-recursive function on its input formula size. Problems such as this one can be only categorized in sub-recursive complexity classes beyond Ackermann, such as the extended Grzegorczyk and Fast-growing complexity hierarchies [Sch16]. These lower bounds follow from the fact that XPath($\rightarrow, \rightarrow^*$) on data words —*i.e.*, where $\rightarrow$ is interpreted as the next-position modality and $\rightarrow^*$ as the future modality— is already non-primitive recursive hard [DL09; FS09]. Notice that in this case the satisfiability on data words is harder than on (unranked) data trees, since the word structure imposes additional data constraints (all data values must appear in the same branch) which makes possible to encode harder problems. In our last section we include more details on these non-primitive recursive lower bounds.

Theorem 2. *[Fig12a] SAT-*XPath($\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$) *is decidable, non-primitive recursive hard.*

The decidability result is obtained by reducing the satisfiability problem to the emptiness problem for a class of register automata. Register automata, introduced in [KF94], are non-deterministic

finite-state automata running on data words and binary data trees, extended with a collection of 'registers' that can store data values and compare them for equality. The alternating version of this model of automata restricted to having only one register was shown to have a decidable emptiness problem in the case of data words [DL09] and data trees [JL11].[2]

The decidability of top-down Alternating Tree 1-Register Automata (ATRA) already allows to easily prove decidability for the satisfiability of a fragment of XPath$(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ [JL11], restricted to data tests of the form $\langle \varepsilon = \alpha \rangle$ and $\langle \varepsilon \neq \alpha \rangle$. This is done by compiling each formula to an ATRA automaton expressing the same property and then testing the automaton for emptiness.[3]

However, formulas of the form $\langle \alpha = \beta \rangle$ are notoriously difficult to encode in a register automata, especially when they are negated. Indeed, note that $\neg\langle \alpha = \beta \rangle$ expresses that the set of data values reachable through α is disjoint from the set of data values reachable through β, this is a *global* property of data values that cannot be expressed in an ATRA model using one register (or any given number for that matter).

In order to tackle $(\neg)\langle \alpha = \beta \rangle$ formulas, this model was extended with the ability of non-deterministically guessing data values, and the possibility to operate simultaneously with all data values seen along the execution (which can be thought of a restricted form of universal quantification on the data values). Decidability for this extension can be shown to be decidable [Fig12a] through a reduction to the coverability problem of a well-structured transition system [FS01]. This extension comes at the cost of the model being no longer closed under complementation; in fact, it can express $\langle \alpha = \beta \rangle$ but not $\neg\langle \alpha = \beta \rangle$ (nor, for instance, the *key* property (1) stated before [Fig12a, Lemma 6.3]). Thus, even the extended ATRA falls short of expressive power to capture forward-XPath! However, these extra features allow to have an effective (though non-trivial) reduction from SAT-XPath$(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ into the emptiness problem for extended ATRA [Fig12a].

As it was the case in the previous section, due to the fact of reducing satisfiability to emptiness of an automata model, the same proof is trivially extended to show that SAT-regXPath$(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ is decidable.

Satisfiability under tree constraints. In the presence of XML constraints, such as XML Schema, DTD's, or in general regular tree constraints, the decidability result is preserved. This is because the ATRA model can encode any regular tree language.

4.3 Vertical fragment

A different extension of the downward fragment of XPath is the *vertical fragment*, where formulas can also navigate upward using the parent and ancestor axes, but they don't use any horizontal navigation: XPath$(\downarrow, \downarrow_*, \uparrow, \uparrow^*)$. This fragment is still decidable but, once again, with non-primitive recursive complexity.

Theorem 3. *[FS17] SAT-*XPath$(\downarrow, \downarrow_*, \uparrow, \uparrow^*)$ *is decidable, non-primitive recursive hard.*

The decidability proof of this fact involves the introduction of a new model of alternating register automata which, contrary of ATRA, is *bottom-up* and runs on *unranked* data trees.

The model of Bottom-up 1-register alternating Data Automata (or BUDA) has one register to store and compare data values in multiple ways: it can compare the stored data value with the current node's value, or with the data value of some (or all) descendant nodes reachable through a path satisfying a given regular property. Hence, in a way it has a two-way behaviour: it moves

[2] As soon as two registers are allowed the emptiness problem becomes undecidable [DL09].

[3] To be precise, ATRA runs over the first-child/next-sibling binary decomposition of the data tree.

upward but it can perform data tests on the subtree that has been already processed. It has also the two extensions introduced for ATRA: the guessing of data values and the universal alternation over all data values in the subtree. As in the previous case, decidability for this model has been shown to be decidable [FS17] through a reduction to the coverability problem of a well-structured transition system, though in this case the compatible well-quasi-order (wqo) acting on the transition system is more involved. Interestingly, vertical-XPath, as well as BUDA enjoy the subtree replication property explained before, which is crucial for showing compatibility of the transition system with respect to the wqo. In fact, the emptiness of BUDA on the class of *ranked* data trees is undecidable; just as the satisfiability of vertical-XPath on ranked data trees is undecidable.

Finally, one can effectively translate any node expression of XPath into a BUDA preserving the expressive power, which yields the decidability of vertical-XPath. As before, the same proof yields decidability for the satisfiability problem of vertical-regXPath.

4.4 Transitive axes

All the lower bounds mentioned insofar use some '1-step' modality (such as parent) and its transitive closure (such as ancestor). But what happens if we don't have 1-step modalities? That is, do any of the results above change if we only have transitive axes? Surprisingly, the answer depends on whether the transitivity is reflexive or not.

Already on data words, with only one transitive axis $\rightarrow^+$ there is a way to encode (in a very weak sense) the one-step axis $\rightarrow$. That is, the possibility to navigate the word and test for equality of data values is expressive enough to reduce SAT-XPath$(\rightarrow, \rightarrow^+)$ into SAT-XPath$(\rightarrow^+)$. Since XPath$(\rightarrow^+)$ is non-primitive recursive on data words, it follows that for any set of axes Σ containing $\rightarrow^+$, ${}^+\!\leftarrow$, or $\uparrow^+$ (that is, some transitive axis in a non-branching direction), we have that XPath(Σ) is either undecidable or decidable with non-primitive recursive complexity [FS09]. Further, if we also have some past navigation, as in XPath$(\rightarrow^+, {}^+\!\leftarrow)$, we lose decidability. In fact, these results summarize the bleak picture of the complexity of the satisfiability of XPath: only $\downarrow_+$ can be used without incurring into enormous complexities.

However, the previous hardness results only work for *transitive* axes, as opposed to *reflexive-transitive* ones. It may come as a surprise that while SAT-XPath$(\rightarrow^+)$ is non-primitive recursive hard on data words, SAT-XPath$(\rightarrow^*)$ is of elementary complexity, and while SAT-XPath$(\rightarrow^+, {}^*\!\leftarrow)$ is undecidable [FS09], SAT-XPath$(\rightarrow^*, {}^*\!\leftarrow)$ is decidable. In fact, both SAT-XPath$(\rightarrow^*)$ and SAT-XPath$(\rightarrow^*, {}^*\!\leftarrow)$ are ExpSpace-complete [Fig11]. What is more, this good behavior extends also to trees, as SAT-XPath$(\rightarrow^*, {}^*\!\leftarrow, \downarrow_*)$ is also of elementary complexity [Fig13] (somewhere between ExpSpace and 3ExpSpace).

The conclusion is that on this kind of path logics on data trees, while transitive axes are very costly, reflexive-transitive axes allow to fully navigate the tree while preserving an elementary complexity. Table 2 summarizes these results.

Satisfiability under tree constraints. In view of the results above, satisfiability under an XML Schema or DTD inherits all the non-primitive recursive bounds known for data words, since one can simply restrict the model to be linear with a DTD or XML Schema. Thus, while SAT-XPath$(\downarrow, \downarrow_*)$ is ExpTime in the absence of constraints, it becomes non-primitive recursive hard in the presence of a DTD, since SAT-XPath$(\rightarrow^+)$ is NPR-hard on data words (it is further decidable since even regXPath$(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ under regular constraints is decidable, as already mentioned). Likewise, while SAT-XPath$(\downarrow, \downarrow_*, \uparrow, \uparrow^*)$ is decidable in the absence of constraints, as soon as we can impose a limit on the number of children of a node it becomes undecidable, by reduction from SAT-

$\downarrow_*$	$\rightarrow^+$	$\rightarrow^*$	${}^*\!\leftarrow$	$\uparrow^+$	**Complexity**
	•				decidable, NPR-hard
				•	decidable, NPR-hard
	•		•		undecidable
•	•			•	undecidable
		•	•		ExpSpace-complete
•		•	•		in 3ExpSpace
•		•		•	*open*

Tab. 2: Satisfiability for XPath with transitive axes. All complexity results hold for regXPath as well. NPR stands for non-primitive recursive complexity.

XPath($\rightarrow^+, {}^+\!\leftarrow$) on data words.

4.5 On very high lower bounds

All the non-primitive recursive lower bounds are shown by reduction from the emptiness problem of *faulty counter automata*. This is a sort of unreliable counter automata (*a.k.a.* Minsky machine) with tests for zero. Remember that in a counter automata counters can be decremented, incremented and tested for zero. While in a counter automata the decrement of a counter with value 0 is not allowed, in the faulty model it is always allowed, the semantics being that a decrement of 0 remains 0. This seemingly trivial alteration allows to regain decidability in the otherwise undecidable model [Min61] of counter automata. It is known [Sch02; Sch10] that the emptiness problem for this class of automata is decidable and not primitive recursive hard, more precisely complete for the Ackermannian class $\mathbf{F}_\omega$ [Sch16].

All the reductions in this setting are centered around the following strategy: A path encodes the accepting run witnessing the non-emptiness of the faulty counter automaton. Labels are used to encode transitions, and data values are used for matching each incrementing transition (*i.e.*, a transition of the form $(q, inc(c_i), q')$) with a later decrementing transition of the same counter (*i.e.*, some $(p, dec(c_i), p')$). Concretely, in the case of data words, the main properties to test are:

1. *each incrementing (resp. decrementing) transition is assigned a unique data value* —a sort of Key property (1)—;
2. *for each incrementing transition of counter c there is a matching future decrementing transition of counter c with the same data value;*
3. *between an increment of c and its matching decrement there cannot be a transition that tests for zero counter c.*

These properties, in addition to the usual properties on finite state automata, ensure that the run is a valid run for the faulty counter system. The key property is the last one, which can be expressed in XPath by expressing that there is no node where a formula of the form $\langle [\varphi_{inc(i)}] = \rightarrow^*[\varphi_{tz(i)}]\rightarrow^*[\varphi_{dec(i)}]\rangle$ holds, assuming φ_a tests whether the current node carries a label with a transition performing the action a. Interestingly, this kind of test cannot be expressed in FO^2 since it would require three variables.

Observe that for each increment there exists at least one later matching decrement before the test for zero, but nothing avoids that there be more than one, which in fact does not matter when working under the faulty semantics. These properties can be encoded on data words using XPath$(\rightarrow, \rightarrow^*)$, which is why XPath$(\rightarrow, \rightarrow^*)$ or XPath$(\uparrow, \uparrow^*)$ are non-primitive recursive hard (note when going upwards in a tree, $\uparrow, \uparrow^*$ behave as in a data word, and similarly for $\rightarrow, \rightarrow^*$).

If we further add the property

4. *for each decrementing transition of counter c there is a previous incrementing transition of counter c with the same data value,*

then we end up encoding non emptiness for 'non-faulty' counter automata —since we now have a bijection between increments and decrements for each counter— and in this way we obtain undecidability. Thus, if the logic can express both future-looking and past-looking data properties of the form above, its satisfiability problem is bound to be undecidable. This is why while SAT-XPath$(\rightarrow, \rightarrow^*)$ is decidable (and non-primitive recursive) on data words, SAT-XPath$(\rightarrow, \rightarrow^*, {}^*\!\leftarrow)$ is not. A notable exception to this rule of thumb is XPath$(\rightarrow^*, {}^*\!\leftarrow)$, and more generally XPath$(\downarrow_*, \rightarrow^*, {}^*\!\leftarrow)$ as discussed in the previous section.

5 Conclusion

Decidability results on XPath are scarce, and most logics on data word with desirable properties (closed under boolean connectives, having the ability to compare distant nodes for data equality) are undecidable. The fragments of XPath we have described here are rather rare exceptions to this landscape of undecidability. These are the downward, forward, and vertical fragments, as well as the fragment with the reflexive-transitive closure of the child, right-sibling and left-sibling.

Acknowledgement Work partially supported by ANR project DELTA, grant ANR-16-CE40-0007.

References

[Abi+13] Serge Abiteboul, Pierre Bourhis, Anca Muscholl, and Zhilin Wu. "Recursive queries on trees and data trees". In: *International Conference on Database Theory (ICDT)*. ACM Press, 2013, pp. 93–104. DOI: 10.1145/2448496.2448509.

[Abr+16] Sergio Abriola, Pablo Barceló, Diego Figueira, and Santiago Figueira. "Bisimulations on Data Graphs". In: *Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 2016, pp. 309–318.

[Abr+17] Sergio Abriola, María Emilia Descotte, Raul Fervari, and Santiago Figueira. "Axiomatizations for downward XPath on data trees". In: *J. Comput. Syst. Sci.* 89 (2017), pp. 209–245. DOI: 10.1016/j.jcss.2017.05.008.

[ACK11] Foto N. Afrati, Sara Cohen, and Gabriel M. Kuper. "On the complexity of tree pattern containment with arithmetic comparisons". In: *Information Processing Letters (IPL)* 111.15 (2011), pp. 754–760. DOI: 10.1016/j.ipl.2011.04.014.

[AF16] Carlos Areces and Raul Fervari. "Hilbert-Style Axiomatization for Hybrid XPath with Data". In: *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings*. Vol. 10021. Lecture Notes in Computer Science. Springer, 2016, pp. 34–48. DOI: 10.1007/978-3-319-48758-8_3.

[AFF17] Sergio Abriola, Diego Figueira, and Santiago Figueira. "Logics of Repeating Values on Data Trees and Branching Counter Systems". In: *International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*. Vol. 10203. Lecture Notes in Computer Science. Springer, 2017, pp. 196–212. DOI: 10.1007/978-3-662-54458-7_12.

[AFS17] Carlos Areces, Raul Fervari, and Nahuel Seiler. "Tableaux for Hybrid XPath with Data". In: *Progress in Artificial Intelligence (EPIA)*. Vol. 10423. Lecture Notes in Computer Science. Springer, 2017, pp. 611–623. DOI: 10.1007/978-3-319-65340-2_50.

[Bár+12] Vince Bárány, Mikołaj Bojańczyk, Diego Figueira, and Paweł Parys. "Decidable classes of documents for XPath". In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*. Leibniz International Proceedings in Informatics (LIPIcs). Leibniz-Zentrum für Informatik, 2012.

[BB07] Henrik Björklund and Mikołaj Bojańczyk. "Bounded Depth Data Trees". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 4596. Lecture Notes in Computer Science. Springer, 2007, pp. 862–874. DOI: 10.1007/978-3-540-73420-8_74.

[BFG08] Michael Benedikt, Wenfei Fan, and Floris Geerts. "XPath satisfiability in the presence of DTDs". In: *Journal of the ACM* 55.2 (2008), pp. 1–79. DOI: 10.1145/1346330.1346333.

[BK08] Michael Benedikt and Christoph Koch. "XPath leashed". In: *ACM Computing Surveys* 41.1 (2008). DOI: 10.1145/1456650.1456653.

[BL10] Mikołaj Bojańczyk and Sławomir Lasota. "An extension of data automata that captures XPath". In: *Annual IEEE Symposium on Logic in Computer Science (LICS)*. 2010.

[BLS16] David Baelde, Simon Lunel, and Sylvain Schmitz. "A Sequent Calculus for a Modal Logic on Finite Data Trees". In: *EACSL Annual Conference on Computer Science Logic (CSL)*. Vol. 62. Leibniz International Proceedings in Informatics (LIPIcs). Leibniz-Zentrum für Informatik, 2016, 32:1–32:16. DOI: 10.4230/LIPIcs.CSL.2016.32.

[Boj+09] Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. "Two-variable logic on data trees and XML reasoning". In: *Journal of the ACM* 56.3 (2009), pp. 1–48. DOI: 10.1145/1516512.1516515.

[Boj+10] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. "Two-variable logic on data words". In: *ACM Transactions on Computational Logic* (2010).

[BP11] Mikołaj Bojańczyk and Paweł Parys. "XPath evaluation in linear time". In: *Journal of the ACM* 58.4 (2011), p. 17.

[CD99] James Clark and Steve DeRose. *XML path language (XPath)*. Website. W3C Recommendation. http://www.w3.org/TR/xpath. 1999.

[Dav+13] Claire David, Amélie Gheerbrant, Leonid Libkin, and Wim Martens. "Containment of pattern-based queries over data trees". In: *International Conference on Database Theory (ICDT)*. ACM Press, 2013, pp. 201–212. DOI: 10.1145/2448496.2448521.

[DDG07] Stéphane Demri, Deepak D'Souza, and Régis Gascon. "Decidable Temporal Logic with Repeating Values". In: *Symposium on Logical Foundations of Computer Science (LFCS)*. Vol. 4514. LNCS. Springer, 2007, pp. 180–194. DOI: 10.1007/978-3-540-72734-7_13.

[DFP16] Stéphane Demri, Diego Figueira, and M. Praveen. "Reasoning about Data Repetitions with Counter Systems". In: *Logical Methods in Computer Science (LMCS)* 12.3 (2016). DOI: 10.2168/LMCS-12(3:1)2016.

[DL09] Stéphane Demri and Ranko Lazić. "LTL with the freeze quantifier and register automata". In: *ACM Transactions on Computational Logic* 10.3 (2009). DOI: 10.1145/1507244.1507246.

[DLN05] Stéphane Demri, Ranko Lazić, and David Nowak. "On the Freeze Quantifier in Constraint LTL: Decidability and Complexity". In: *International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE Computer Society Press, 2005, pp. 113–121. DOI: 10.1016/j.ic.2006.08.003.

[DLT12] Claire David, Leonid Libkin, and Tony Tan. "Efficient reasoning about data trees via integer linear programming". In: *ACM Transactions on Database Systems (TODS)* 37.3 (2012), p. 19. DOI: 10.1145/2338626.2338632.

[DT01] Alin Deutsch and Val Tannen. "Containment and Integrity Constraints for XPath". In: *Proceedings of the International Workshop on Knowledge Representation meets Databases (KRDB))*. Vol. 45. CEUR Workshop Proceedings. CEUR-WS.org, 2001.

[FCG04] Wenfei Fan, Chee Yong Chan, and Minos N. Garofalakis. "Secure XML Querying with Security Views". In: *ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM Press, 2004, pp. 587–598.

[FFA15] Diego Figueira, Santiago Figueira, and Carlos Areces. "Model Theory of XPath on Data Trees. Part I: Bisimulation and Characterization". In: *Journal of Artificial Intelligence Research (JAIR)* 53 (2015), pp. 271–314. DOI: 10.1613/jair.4658.

[Fig+17] Diego Figueira, Ranko Lazic, Jérôme Leroux, Filip Mazowiecki, and Grégoire Sutre. "Polynomial-Space Completeness of Reachability for Succinct Branching VASS in Dimension One". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 80. Leibniz International Proceedings in Informatics (LIPIcs). Leibniz-Zentrum für Informatik, 2017, 119:1–119:14. DOI: 10.4230/LIPIcs.ICALP.2017.119.

[Fig10] Diego Figueira. "Forward-XPath and extended register automata on data-trees". In: *International Conference on Database Theory (ICDT)*. ACM Press, 2010. DOI: 10.1145/1804669.1804699.

[Fig11] Diego Figueira. "A decidable two-way logic on data words". In: *Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, 2011, pp. 365–374. DOI: 10.1109/LICS.2011.18.

[Fig12a] Diego Figueira. "Alternating Register Automata on Finite Data Words and Trees". In: *Logical Methods in Computer Science (LMCS)* 8.1 (2012). DOI: 10.2168/LMCS-8(1:22)2012.

[Fig12b] Diego Figueira. "Decidability of Downward XPath". In: *ACM Transactions on Computational Logic* 13.4 (2012). DOI: 10.1145/2362355.2362362.

[Fig13] Diego Figueira. "On XPath with transitive axes and data tests". In: *ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, 2013, pp. 249–260. DOI: 10.1145/2463664.2463675.

[FL14] Diego Figueira and Leonid Libkin. "Pattern logics and auxiliary relations". In: *Annual IEEE Symposium on Logic in Computer Science (LICS)*. ACM Press, 2014, 40:1–40:10. DOI: 10.1145/2603088.2603136.

[FS01] Alain Finkel and Philippe Schnoebelen. "Well-Structured Transition Systems Everywhere!" In: *Theoretical Computer Science* 256.1-2 (2001), pp. 63–92. DOI: 10.1016/S0304-3975(00)00102-X.

[FS09] Diego Figueira and Luc Segoufin. "Future-looking logics on data words and trees". In: *International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Vol. 5734. LNCS. Springer, 2009, pp. 331–343. DOI: 10.1007/978-3-642-03816-7_29.

[FS17] Diego Figueira and Luc Segoufin. "Bottom-up automata on data trees and Vertical XPath". In: *Logical Methods in Computer Science (LMCS)* 13.4 (2017). DOI: 10.23638/LMCS-13(4:5)2017.

[GF05] Floris Geerts and Wenfei Fan. "Satisfiability of XPath Queries with Sibling Axes". In: *International Symposium on Database Programming Languages (DBPL)*. Vol. 3774. Lecture Notes in Computer Science. Springer, 2005, pp. 122–137. DOI: 10.1007/11601524_8.

[GKP05] Georg Gottlob, Christoph Koch, and Reinhard Pichler. "Efficient algorithms for processing XPath queries". In: *ACM Transactions on Database Systems (TODS)* 30.2 (2005), pp. 444–491. DOI: 10.1145/1071610.1071614.

[Göl+16] Stefan Göller, Christoph Haase, Ranko Lazić, and Patrick Totzke. "A Polynomial-Time Algorithm for Reachability in Branching VASS in Dimension One". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 55. Leibniz International Proceedings in Informatics (LIPIcs). Leibniz-Zentrum für Informatik, 2016, 105:1–105:13. DOI: 10.4230/LIPIcs.ICALP.2016.105.

[JL07] Marcin Jurdziński and Ranko Lazić. "Alternation-free modal mu-calculus for data trees". In: *Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, 2007, pp. 131–140. DOI: 10.1109/LICS.2007.11.

[JL11] Marcin Jurdziński and Ranko Lazić. "Alternating automata on data trees and XPath satisfiability". In: *ACM Transactions on Computational Logic* 12.3 (2011), p. 19. DOI: 10.1145/1929954.1929956.

[KF94] Michael Kaminski and Nissim Francez. "Finite-Memory Automata". In: *Theoretical Computer Science* 134.2 (1994), pp. 329–363. DOI: 10.1016/0304-3975(94)90242-9.

[KSZ10] Ahmet Kara, Thomas Schwentick, and Thomas Zeume. "Temporal Logics on Words with Multiple Data Values". In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*. Leibniz International Proceedings in Informatics (LIPIcs). Leibniz-Zentrum für Informatik, 2010, pp. 481–492. DOI: 10.4230/LIPIcs.FSTTCS.2010.481.

[KT08] Michael Kaminski and Tony Tan. "Tree Automata over Infinite Alphabets". In: *Pillars of Computer Science*. Vol. 4800. Lecture Notes in Computer Science. Springer, 2008, pp. 386–423. DOI: 10.1007/978-3-540-78127-1_21.

[Min61] M. L. Minsky. "Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing Machines". In: *Annals of Mathematics* 74 (1961), pp. 437–455.

[MN07] Wim Martens and Frank Neven. "Frontiers of tractability for typechecking simple XML transformations". In: *J. Comput. Syst. Sci.* 73.3 (2007), pp. 362–390.

[NS06] Frank Neven and Thomas Schwentick. "On the complexity of XPath containment in the presence of disjunction, DTDs, and variables". In: *Logical Methods in Computer Science (LMCS)* 2.3 (2006). DOI: 10.2168/LMCS-2(3:1)2006.

[NSV04] Frank Neven, Thomas Schwentick, and Victor Vianu. "Finite state machines for strings over infinite alphabets". In: *ACM Transactions on Computational Logic* 5.3 (2004), pp. 403–435. DOI: 10.1145/1013560.1013562.

[Sch02] Philippe Schnoebelen. "Verifying Lossy Channel Systems has Nonprimitive Recursive Complexity". In: *Information Processing Letters (IPL)* 83.5 (2002), pp. 251–261. DOI: 10.1016/S0020-0190(01)00337-4.

[Sch10] Philippe Schnoebelen. "Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets". In: *International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Vol. 6281. Lecture Notes in Computer Science. Springer, 2010, pp. 616–628. DOI: 10.1007/978-3-642-15155-2_54.

[Sch16] Sylvain Schmitz. "Complexity Hierarchies beyond Elementary". In: *TOCT* 8.1 (2016), 3:1–3:36. DOI: 10.1145/2858784.

[Tan12] Tony Tan. "An Automata Model for Trees with Ordered Data Values". In: *Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, 2012, pp. 586–595. DOI: 10.1109/LICS.2012.69.